Lecture 10 – Classes & Objects II
https://cw.fel.cvut.cz/wiki/courses/be5b33prg/start

Milan Nemy
Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics
https://beat.ciirc.cvut.cz/people/milan-nemy/
milan.nemy@cvut.cz
OOP is about changing the perspective

- **Syntax for a function call**: `function_name(variable)`
  *function* is the one who executes on the variable

- **Syntax in OOP**: `object_name.function_name()`
  *object* is the one who executes its method on given data / attribute
• Advantage of using a class (e.g. Point) rather than a tuple is that **class methods are sensible operations** for points, but may not be appropriate for other tuples (e.g. calculate the distance from the origin)

• Class allows to **group together sensible operations** as well as data to apply the methods on

• Each instance of the class has its **own state**

• Method **behaves like a function** but it is invoked on a specific instance

---

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_1.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_1.html)
@classmethod

• In the same way class attributes are defined, which are shared between all instances of a class, class methods are defined using `@classmethod` decorator for ordinary method

• Class method still has its calling object as the first parameter, but by convention it is `cls` instead of `self`

• If class method is called from an instance, this parameter will contain the instance object, but if it is called from the class it will contain the class object

• Naming the parameter `cls` serves as reminder that it is not guaranteed to have any instance attributes
What are class methods good for?

• For tasks associated with a class utilizing constants and other class attributes without the need to create any class instances

• **EXAMPLE:** when we write classes to group related constants together with functions which act on them – no need to instantiate these classes at all
class Inst:
    
def __init__(self, name):
        self.name = name
    
def introduce(self):
        print("Hello, I am %s, and my name is " %(self, self.name))

myinst = Inst("Test Instance")
otherinst = Inst("An other instance")
myinst.introduce()
# outputs: Hello, I am <Inst object at x>, and my name is Test Instance
otherinst.introduce()
# outputs: Hello, I am <Inst object at y>, and my name is An other instance

SOURCE https://stackoverflow.com/questions/17134653/difference-between-class-and-instance-methods
class Cls:

    @classmethod
    def introduce(cls):
        print("Hello, I am \%s!" %cls)

Cls.introduce() # same as Cls.introduce(Cls)
# outputs: Hello, I am <class 'Cls'>

Notice that again Cls is passed hiddenly, so we could also say Cls.introduce(Inst) and get output "Hello, I am <class 'Inst'>. This is particularly useful when we're inheriting a class from Cls:

class SubCls(Cls):
    pass

SubCls.introduce()
# outputs: Hello, I am <class 'SubCls'>

SOURCE https://stackoverflow.com/questions/17134653/difference-between-class-and-instance-methods
@staticmethod

- Static method does not have the calling object passed into it as the first parameter

- Static method does not have access to the rest of the class or instance

- Static method is most commonly called from class objects (like class methods)
RECAP: EXAMPLE – STATIC METHODS

```python
class Person:
    TITLES = ('Dr', 'Mr', 'Mrs', 'Ms')

    def __init__(self, name, surname):
        self.name = name
        self.surname = surname

    def fullname(self):  # instance method
        # instance object accessible through self
        return "%s %s" % (self.name, self.surname)

@classmethod
def allowed_titles_starting_with(cls, startswith):  # class method
    # class or instance object accessible through cls
    return [t for t in cls.TITLES if t.startswith(startswith)]

@staticmethod
def allowed_titles_ending_with(endswith):  # static method
    # no parameter for class or instance object
    # we have to use Person directly
    return [t for t in Person.TITLES if t.endswith(endswith)]

In[3]: jane = Person("Jane", "Smith")
In[4]: print(jane.fullname())
Jane Smith
In[5]: print(jane.allowed_titles_starting_with("M"))
['Mr', 'Mrs', 'Ms']
In[6]: print(Person.allowed_titles_starting_with("M"))
['Mr', 'Mrs', 'Ms']
In[7]: print(jane.allowed_titles_ending_with("s"))
['Mrs', 'Ms']
In[8]: print(Person.allowed_titles_ending_with("s"))
['Mrs', 'Ms']
```

SOURCE: http://python-textbook.readthedocs.io/en/1.0/Classes.html# UNDER CC BY-SA 4.0 licence Revision 8e685e710775
• Assume a **rectangle** that is oriented either vertically or horizontally, never at an angle;
• Specify the **upper-left corner** of the rectangle, and its **size**
To specify the upper-left corner embed a **Point object** within the new **Rectangle object**

Create two new Rectangle objects, and then print them
• The dot operator composes.
• The expression `box.corner.x` means:

  “Go to the object that box refers to and select its attribute named corner, then go to that object and select its attribute named x”
• Change the state of an object by making an assignment to one of its attributes

```python
box.width += 50
box.height += 100
```
• *If two objects are the same, does it mean they contain the same data or that they are the same object?*

• The *is* operator was used in previous examples on the lists when explaining aliases: *it allows us to find out if two references refer to the same object*.

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
• **Shallow copy**: is defined as constructing a new collection object and then populating it with references to the child objects found in the original, i.e. a shallow copy is only one level deep. The copying process does not recurse and therefore will not create copies of the child objects.

• **Deep copy**: is defined as recursive copying process, i.e. first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. *Copying an object this way walks the whole object tree to create a fully independent clone of the original object and all of its children.*

source [https://realpython.com/copying-python-objects/](https://realpython.com/copying-python-objects/)
OBJECT EQUALITY

```python
def same_coordinates(p1, p2):
    return (p1.x == p2.x) and (p1.y == p2.y)
```

```python
>>> p1 = Point(3, 4)
>>> p2 = Point(3, 4)
>>> same_coordinates(p1, p2)
True
```

- **Shallow equality**: When `is` is `True`, this type of equality is *shallow equality* because it compares only the *references* and not the *contents* of the objects.

- **Deep equality**: To compare the *contents* of the objects a function like `same_coordinates` needs to be created.

- **IMPORTANT**: If two variables refer to the same object, they have both shallow and deep equality.

Source: [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
Think about **shallow** & **deep** copy when designing classes!

Even though the two lists (or tuples, etc.) are *distinct objects with different memory addresses*, for **lists** the `==` operator tests for *deep equality*, while in the case of **objects** (points) it makes a *shallow test*.

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
• Aliasing makes code difficult to read – *changes made in one place might have unexpected effects in another place*

• Copying object is an **alternative to aliasing**: the **copy module** contains a function copy that can duplicate any object

• **EXAMPLE**: *To copy objects import the copy module and use the copy function to make a new Point: p1 and p2 are not the same point, but they contain the same data (shallow copy)*

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
• Assume **Rectangle**, which contains a reference to a **Point**: copy copies the reference to the **Point** object, so both the old **Rectangle** and the new one refer to the same **Point**

• **Invoking** `grow` on one of the Rectangle objects would not affect the other,

• **Invoking** `move` on either would affect both since **shallow copy has created an alias to the Point that represents the corner**

• Copy module contains a function named **deepcopy** that copies not only the object but also any embedded objects

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
• **Deep copy**: To copy the contents of an object as well as any embedded objects, and any objects embedded in them, etc. *(implemented as deepcopy function in copy module)*

• **Deep equality**: Equality of values, or two references that point to objects that have the same value

• **Shallow copy**: To copy the contents of an object, including any references to embedded objects (one level copy). *(implemented by the copy function in the copy module)*

• **Shallow equality**: Equality of references, or two references that point to the same object

source [http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html](http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html)
• **EXAMPLE:** User-defined type called *MyTime* that records the time of day

• Initializer using an `__init__` method to ensure that every instance is created with appropriate attributes

```python
class MyTime:
    def __init__(self, hrs=0, mins=0, secs=0):
        """ Create a MyTime object initialized to hrs, mins, secs """
        self.hours = hrs
        self.minutes = mins
        self.seconds = secs

tim1 = MyTime(11, 59, 30)
```

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
**EXAMPLE:** Create two `MyTime` objects

- `current_time`, which contains the current time
- `bread_time`, which contains the amount of time it takes for a breadmaker to make bread
- use `add_time` to figure out when the bread will be done

**PROBLEM:** ??

```python
def add_time(t1, t2):
    h = t1.hours + t2.hours
    m = t1.minutes + t2.minutes
    s = t1.seconds + t2.seconds
    sum_t = MyTime(h, m, s)
    return sum_t

>>> current_time = MyTime(9, 14, 30)
>>> bread_time = MyTime(3, 35, 0)
>>> done_time = add_time(current_time, bread_time)
>>> print(done_time)
12:49:30
```

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
EXAMPLE: Create two MyTime objects:
- current_time, which contains the current time
- bread_time, which contains the amount of time it takes for a breadmaker to make bread
- use add_time to figure out when the bread will be done

PROBLEM: We do not deal with cases where the number of seconds or minutes adds up to more than sixty!

```
def add_time(t1, t2):
    h = t1.hours + t2.hours
    m = t1.minutes + t2.minutes
    s = t1.seconds + t2.seconds
    sum_t = MyTime(h, m, s)
    return sum_t
```

```python
>>> current_time = MyTime(9, 14, 30)
>>> bread_time = MyTime(3, 35, 0)
>>> done_time = add_time(current_time, bread_time)
>>> print(done_time)
12:49:30
```
There can be two versions of a function \texttt{add\_time}, pure function or the \texttt{modifier}, which both calculate the sum of two \texttt{MyTime} objects.

Function that creates a new \texttt{MyTime} object and returns a reference to the new object is a \texttt{pure function} because it does not modify any of the objects passed to it as parameters and it has no side effects.

```python
def add_time(t1, t2):
    h = t1.hours + t2.hours
    m = t1.minutes + t2.minutes
    s = t1.seconds + t2.seconds
    sum_t = MyTime(h, m, s)
    return sum_t
```
EXAMPLE MYTIME – PURE FUNCTIONS

```python
#def add_time(t1, t2):

def add_time(t1, t2):
    h = t1.hours + t2.hours
    m = t1.minutes + t2.minutes
    s = t1.seconds + t2.seconds

    if s >= 60:
        s -= 60
        m += 1

    if m >= 60:
        m -= 60
        h += 1

    sum_t = MyTime(h, m, s)
    return sum_t
```

PROBLEM: Do we now deal with cases where the number of seconds or minutes adds up to more than sixty?

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html

22/12/23
Milan Nemy, Czech Technical University in Prague
• It can be useful for a function to modify one or more of the objects it gets as parameters

• Usually, the caller keeps a reference to the objects it passes, so any changes the function are visible to the caller (modifier)

• Function `increment`, which adds a given number of seconds to a MyTime object, is a natural example of a modifier

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• **SOLUTION**: Include functions that work with MyTime objects into the MyTime class, i.e. conversion of the function `increment` to a method

• This conversion means moving the definition into the class and changing the name of the first parameter to `self`

```python
class MyTime:
    # Previous method definitions here...

def increment(self, seconds):
    self.seconds += seconds

    while self.seconds >= 60:
        self.seconds -= 60
        self.minutes += 1

    while self.minutes >= 60:
        self.minutes -= 60
        self.hours += 1

    # Example usage:
    current_time.increment(500)
```

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• **INSIGHT:** MyTime object is actually a **three-digit number in base 60**!

• Another approach —convert the MyTime object into a single number instead, i.e. the method `to_seconds` can be added to the MyTime class to convert any instance into corresponding number of seconds

```python
class MyTime:
    # ...

    def to_seconds(self):
        """Return the number of seconds represented by this instance"""
        return self.hours * 3600 + self.minutes * 60 + self.seconds

hrs = tsecs // 3600
leftoversecs = tsecs % 3600
mins = leftoversecs // 60
secs = leftoversecs % 60
```

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)

Milan Nemy, Czech Technical University in Prague
In OOP we wrap together the **data** and the **operations**

Solution is to rewrite the class initializer so that it can cope with initial values of seconds or minutes that are **outside the range of the normalized values**

(normalized time: 3 hours 12 minutes and 20 seconds; the same time but not normalized 2 hours 70 minutes and 140 seconds)

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
The function `after` can be defined to compare two times and specify whether the first time is strictly after the second.

This solution is a bit more complicated because it operates on two `MyTime` objects not just one.

```python
>>> t1 = MyTime(10, 55, 12)
>>> t2 = MyTime(10, 48, 22)
>>> after(t1, t2)  # Is t1 after t2?
True
```
```python
class MyTime:
    # Previous method definitions here...

def after(self, time2):
    """ Return True if I am strictly greater than time2 """
    if self.hours > time2.hours:
        return True
    if self.hours < time2.hours:
        return False

    if self.minutes > time2.minutes:
        return True
    if self.minutes < time2.minutes:
        return False
    if self.seconds > time2.seconds:
        return True

    return False
```

```python
if current_time.after(done_time):
    print("The bread will be done before it starts!")
```

- **Lines 11-18** will only be reached if the two hour fields are the same.
- The test at **line 16** is only executed if both times have the same hours and the same minutes.

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
The whole example can be made easier using the previously discovered insight of converting the time into single integer!

This is a great way to code this:

If we want to tell if the first time is after the second time, turn them both into integers and compare the integers.

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• **Operator overloading**: *opens the possibility to have different meanings for the same operator when applied to different types*

• **EXAMPLE**: *the `+` in Python means quite different things for integers* (*addition*) *and for strings* (*concatenation*)!  

• To override the addition operator `+` provide a method named `__add__`

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)

```python
class MyTime:
    # Previously defined methods here...

    def __add__(self, other):
        return MyTime(0, 0, self.to_seconds() + other.to_seconds())
```
class MyTime:
    # Previously defined methods here...

def __add__(self, other):
    return MyTime(0, 0, self.to_seconds() + other.to_seconds())

>>> t1 = MyTime(1, 15, 42)
>>> t2 = MyTime(3, 50, 30)
>>> t3 = t1 + t2
>>> print(t3)
05:06:12

• First parameter is the object on which the method is invoked
• Second parameter is named other to distinguish it from self
• To add two MyTime objects create and return a new MyTime object that contains their sum

• The expression t1 + t2 is equivalent to t1.__add__(t2)

source http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html
• **EXAMPLE**: back to the Point class – *adding two points adds their respective (x, y) coordinates*

• There are several ways to override the behavior of the multiplication operator by defining a method named `__mul__` or `__rmul__` or both

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
If the left operand of * is a **Point**, Python invokes **__mul__**, which assumes that the other operand is also a Point (this computes the **dot product** of the two Points)

If the left operand of * is a **primitive type** and the right operand is a **Point**, Python invokes **__rmul__**, which performs **scalar multiplication**

The result is always a new **Point** whose coordinates are a multiple of the original coordinates

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
OPERATOR OVERLOADING

PROBLEM: How is \( p2 \times 2 \) evaluated?

```python
>>> p1 = Point(3, 4)
>>> p2 = Point(5, 7)
>>> print(p1 * p2)
43
>>> print(2 * p2)
(10, 14)
```

source: http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html
• **PROBLEM**: How is \( p2 \times 2 \) evaluated?

• Since the first parameter is a Point, Python invokes `__mul__` with 2 as the second argument

• Inside `__mul__`, the program tries to access the \( x \) coordinate of `other`, which fails because an integer has no attributes
• **EXAMPLE:** `front_and_back` – consider a function which prints a list twice: forward and backward

• The reverse method is a **modifier** therefore a copy needs to be made before applying it (this way we prevent to modify the list the function gets as a parameter!)

• Function that can take arguments with different types and handles them accordingly is called **polymorphic**

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• Polymorphism == ability to process objects differently based on data type

• There are certain operations that can be applied to many types, such as the arithmetic operations ...

• EXAMPLE: The multadd operation takes three parameters: multiplies the first two and then adds the third

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• **EXAMPLE:** *The multadd operation takes three parameters: multiplies the first two and then adds the third*

• **The first case:** the Point is multiplied by a scalar and then added to another Point

• **The second case:** the dot product yields a numeric value, so the third parameter also has to be a numeric value

source [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
• **Python’s fundamental rule of polymorphism** is called the **duck typing rule**: *If all of the operations inside the function can be applied to the type, the function can be applied to the type.*

• Operations in the **front_and_back**: *copy, reverse, print*

• **EXAMPLE**: What about our Point class? The **copy** method works on any object; already written a **__str__** method for Point objects for the **str()** conversion, only the **reverse** method for the Point class is needed!

source: [http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html](http://openbookproject.net/thinkcs/python/english3e/even_more_oop.html)
REFERENCES

This lecture re-uses selected parts of the OPEN BOOK PROJECT
Learning with Python 3 (RLE)
http://openbookproject.net/thinkcs/python/english3e/index.html
available under GNU Free Documentation License Version 1.3)

• Version date: October 2012
• by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers
  (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
• Source repository is at  https://code.launchpad.net/~thinkcspy-rle-
team/thinkcspy/thinkcspy3-rle
• For offline use, download a zip file of the html or a pdf version
  from  http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/

This lecture re-uses selected parts of the PYTHON TEXTBOOK
Object-Oriented Programming in Python
http://python-textbok.readthedocs.io/en/1.0/Classes.html#
(released under CC BY-SA 4.0 licence Revision 8e685e710775)